RDF AND SPARQL

Part V: Semantics of SPARQL

Sebastian Rudolph
ICCL Summer School

Dresden, August 2013
Agenda

1. Recap
2. SPARQL Semantics
3. Transformation of Queries into Algebra Objects
4. Evaluation of the SPARQL Algebra
5. Operators for the Modifiers
6. Summary
Agenda

1. Recap
2. SPARQL Semantics
3. Transformation of Queries into Algebra Objects
4. Evaluation of the SPARQL Algebra
5. Operators for the Modifiers
6. Summary
**Recap: Introduced SPARQL Features**

<table>
<thead>
<tr>
<th>Basic Structure</th>
<th>Filter</th>
<th>Modifiers</th>
<th>Output Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFIX</td>
<td>BOUND</td>
<td>ORDER BY</td>
<td>SELECT</td>
</tr>
<tr>
<td>WHERE</td>
<td>isURI</td>
<td>LIMIT</td>
<td>CONSTRUCT</td>
</tr>
</tbody>
</table>

**Graph Patterns**

<table>
<thead>
<tr>
<th>Basic Graph Patterns</th>
<th>Filter</th>
<th>Modifiers</th>
<th>Output Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOUND</td>
<td>ORDER BY</td>
<td>SELECT</td>
</tr>
<tr>
<td></td>
<td>isURI</td>
<td>LIMIT</td>
<td>CONSTRUCT</td>
</tr>
<tr>
<td>{...}</td>
<td>isBLANK</td>
<td>OFFSET</td>
<td>ASK</td>
</tr>
<tr>
<td>OPTIONAL</td>
<td></td>
<td>DISTINCT</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>UNION</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filter</th>
<th>Modifiers</th>
<th>Output Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>STR</td>
<td>ORDER BY</td>
<td>SELECT</td>
</tr>
<tr>
<td>LANG</td>
<td>LIMIT</td>
<td>CONSTRUCT</td>
</tr>
<tr>
<td>DATATYPE</td>
<td>OFFSET</td>
<td>ASK</td>
</tr>
<tr>
<td>sameTERM</td>
<td>DISTINCT</td>
<td>DESCRIBE</td>
</tr>
<tr>
<td>langMATCHES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REGEX</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

1. Recap
2. SPARQL Semantics
3. Transformation of Queries into Algebra Objects
4. Evaluation of the SPARQL Algebra
5. Operators for the Modifiers
6. Summary
Semantics of Query Languages

So far only informal presentation of SPARQL features

- User: “Which answers can I expect for my query?”
- Developer: “Which behaviour is expected from my SPARQL implementation?”
- Marketing: “Is our product already conformant with the SPARQL standard?”

⇝ Formal semantics should clarify these questions . . .
Semantics of Query Languages (1)

Query Entailment

- Query as description of allowed results
- Data as set of logical assumptions (axiom set/theory)
- Results as logical entailment

⇒ OWL DL and RDF(S) as query languages
⇒ conjunctive queries
Semantics of Query Languages (2)

Query Algebra

- Query as instruction for computing the results
- Queried data as input
- Results as output

⇝ Relational algebra for SQL
⇝ SPARQL Algebra
Agenda

1. Recap
2. SPARQL Semantics
3. Transformation of Queries into Algebra Objects
4. Evaluation of the SPARQL Algebra
5. Operators for the Modifiers
6. Summary
Translation into SPARQL Algebra

```sparql
  FILTER (?price < 15)
  OPTIONAL { ?book ex:title ?title } 
  { ?book ex:author ex:Shakespeare } UNION 
  { ?book ex:author ex:Marlowe } 
}
```

Semantics of a SPARQL query:

1. Transformation of the query into an algebra expression
2. Evaluation of the algebra expression
Translation into SPARQL Algebra

```sparql
{ ?book ex:price ?price
  FILTER (?price < 15)
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare } UNION
  { ?book ex:author ex:Marlowe }
}
```

Attention: Filters apply to the whole group in which they occur
Translation into SPARQL Algebra

```
{ ?book ex:price ?price
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare } UNION
  { ?book ex:author ex:Marlowe }
  FILTER (?price < 15)
}
```

1. Expand abbreviated IRIs
Translation into SPARQL Algebra

```sparql
    <http://ex.org/Shakespeare> } UNION 
    <http://ex.org/Marlowe> } 
  FILTER (?price < 15)
}
```
Translation into SPARQL Algebra

    <http://ex.org/Shakespeare> } UNION 
    <http://ex.org/Marlowe> } 
  FILTER (?price < 15)
}

2. Replace triple patterns with operator Bgp(·)
Translation into SPARQL Algebra

\[
\{ \text{Bgp}(\text{?book} <\text{http://ex.org/price}> \ ?\text{price}) \\
\quad \text{OPTIONAL} \{ \text{Bgp}(\text{?book} <\text{http://ex.org/title}> \ ?\text{title}) \} \\
\quad \{ \text{Bgp}(\text{?book} <\text{http://ex.org/author}> \\
\quad \quad <\text{http://ex.org/Shakespeare}>)) \} \quad \text{UNION} \\
\quad \{ \text{Bgp}(\text{?book} <\text{http://ex.org/author}> \\
\quad \quad <\text{http://ex.org/Marlowe}>)) \} \\
\quad \text{FILTER (}\ ?\text{price} < 15) \\
\}\]

Translation into SPARQL Algebra

\{
\text{Bgp}(\text{?book} \: \text{<http://ex.org/price>} \: \text{?price})
\text{OPTIONAL } \{\text{Bgp}(\text{?book} \: \text{<http://ex.org/title>} \: \text{?title})\}
\{\text{Bgp}(\text{?book} \: \text{<http://ex.org/author>}
               \text{<http://ex.org/Shakespeare>})\}\text{ UNION }
\{\text{Bgp}(\text{?book} \: \text{<http://ex.org/author>}
               \text{<http://ex.org/Marlowe>})\}\text{ FILTER }(?\text{price} < 15)
\} \\

3. Introduce the LeftJoin(·) operator for optional parts
Translation into SPARQL Algebra

\[
\{ \text{LeftJoin} (\text{Bgp}(?book <\text{http://ex.org/price}> ?price), \\
\text{Bgp}(?book <\text{http://ex.org/title}> ?title), \\
\text{true}) \\
\{\text{Bgp}(?book <\text{http://ex.org/author}> \\
\text{<http://ex.org/Shakespeare>})\} \text{ UNION} \\
\{\text{Bgp}(?book <\text{http://ex.org/author}> \\
\text{<http://ex.org/Marlowe>})\} \\
\text{FILTER} (\text{?price < 15}) \\
\}
\]
Translation into SPARQL Algebra

{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
            true)
    {Bgp(?book <http://ex.org/author>
          <http://ex.org/Shakespeare>)} UNION
    {Bgp(?book <http://ex.org/author>
          <http://ex.org/Marlowe>)}
    FILTER (?price < 15)
}

4. Combine alternative graph patterns with Union(·) operator
   ~~~ Refers to neighbouring patterns and has higher precedence than conjunction (left associative)
Translation into SPARQL Algebra

```sparql
{  
  LeftJoin(Bgp(?book <http://ex.org/price> ?price),
            true)
  Union(Bgp(?book <http://ex.org/author>
               <http://ex.org/Shakespeare>),
        Bgp(?book <http://ex.org/author>
             <http://ex.org/Marlowe>))
  FILTER (?price < 15)
}
```
Translation into SPARQL Algebra

{  LeftJoin(Bgp(?book <http://ex.org/price> ?price),
   true)
   Union(Bgp(?book <http://ex.org/author>
   <http://ex.org/Shakespeare>),
   Bgp(?book <http://ex.org/author>
   <http://ex.org/Marlowe>))
   FILTER (?price < 15)
}

5. Apply Join(·) operator to join non-filter elements
Translation into SPARQL Algebra

{  
  Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
            true),
    Union(Bgp(?book <http://ex.org/author>
            <http://ex.org/Shakespeare>),
            Bgp(?book <http://ex.org/author>
            <http://ex.org/Marlowe>)))
  FILTER (?price < 15)
}

Translation into SPARQL Algebra

\[
\{ \text{Join(}
\text{LeftJoin(Bgp(} \text{?book <http://ex.org/price> ?price) ,)
\text{Bgp(} \text{?book <http://ex.org/title> ?title) ,)
\text{true) ,}
\text{Union(Bgp(} \text{?book <http://ex.org/author>}
\text{<http://ex.org/Shakespeare>),}
\text{Bgp(} \text{?book <http://ex.org/author>}
\text{<http://ex.org/Marlowe>))})
\text{FILTER (} \text{?price < 15}\text{)}
\text{)}\}
\]

6. Translate a group with filters with the Filter(·) operator
Translation into SPARQL Algebra

\[
\text{Filter}(\text{?price} < 15, \\
\text{Join(} \\
\quad \text{LeftJoin}(\text{Bgp(?book <http://ex.org/price> ?price),} \\
\quad \quad \text{Bgp(?book <http://ex.org/title> ?title),} \\
\quad \quad \text{true}), \\
\text{Union}(\text{Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),} \\
\quad \quad \text{Bgp(?book <http://ex.org/author> <http://ex.org/Marlowe>))})}
\]
Translation into SPARQL Algebra

Filter(?price < 15, 
    Join( 
        LeftJoin(Bgp(?book <http://ex.org/price> ?price), 
            true), 
        Union(Bgp(?book <http://ex.org/author> 
            <http://ex.org/Shakespeare>), 
            Bgp(?book <http://ex.org/author> 
            <http://ex.org/Marlowe>))))

• Online translation tool: 
  http://sparql.org/query-validator.html
Agenda

1. Recap
2. SPARQL Semantics
3. Transformation of Queries into Algebra Objects
4. Evaluation of the SPARQL Algebra
5. Operators for the Modifiers
6. Summary
Semantics of the SPARQL Algebra Operations

Now we have an algebra object, but what do the algebra operations mean?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bgp($P$)</td>
<td>match/evaluate pattern $P$</td>
</tr>
<tr>
<td>Join($M_1$, $M_2$)</td>
<td>conjunctive join of solutions $M_1$ and $M_2$</td>
</tr>
<tr>
<td>Union($M_1$, $M_2$)</td>
<td>union of solutions $M_1$ with $M_2$</td>
</tr>
<tr>
<td>LeftJoin($M_1$, $M_2$, $F$)</td>
<td>optional join of $M_1$ with $M_2$ with filter constraint $F$ (true if no filter given)</td>
</tr>
<tr>
<td>Filter($F$, $M$)</td>
<td>filter solutions $M$ with constraint $F$</td>
</tr>
<tr>
<td>$Z$</td>
<td>empty pattern (identity for join)</td>
</tr>
</tbody>
</table>
Now we have an algebra object, but what do the algebra operations mean?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bgp(P)</td>
<td>match/evaluate pattern P</td>
</tr>
<tr>
<td>Join(M₁, M₂)</td>
<td>conjunctive join of solutions M₁ and M₂</td>
</tr>
<tr>
<td>Union(M₁, M₂)</td>
<td>union of solutions M₁ with M₂</td>
</tr>
<tr>
<td>LeftJoin(M₁, M₂, F)</td>
<td>optional join of M₁ with M₂ with filter constraint F (true if no filter given)</td>
</tr>
<tr>
<td>Filter(F, M)</td>
<td>filter solutions M with constraint F</td>
</tr>
<tr>
<td>Z</td>
<td>empty pattern (identity for join)</td>
</tr>
</tbody>
</table>

- Only Bgp(·) matches or evaluates graph patterns
- We can use entailment checking rather than graph matching
Definition of the SPARQL Operators

How can we define that more formally?

Output:
• “solution set” (formatting irrelevant)

Input:
• Queried (active) graph
• Partial results from previous evaluation steps
• Different parameters according to the operation

How can we formally describe the “results”?
SPARQL Results

Intuition: Results coded as tables of variable assignments

Result:

List of solutions (solution sequence)

→ each solution corresponds to one table row
SPARQL Results

Intuition: Results coded as tables of variable assignments

Result:
List of solutions (solution sequence)

⇝ each solution corresponds to one table row

Solution:
Partial function
  • Domain: relevant variables
  • Range: IRIs $\cup$ blank nodes $\cup$ RDF literals

⇝ Unbound variables are those that have no assigned value (partial function)
Let \( P \) be a basic graph pattern. A partial function \( \mu \) is a solution for \( \text{Bgp}(P) \) over the queried (active) graph \( G \) if:

1. the domain of \( \mu \) is exactly the set of variables in \( P \),
2. there exists an assignment \( \sigma \) from blank nodes in \( P \) to IRIs, blank nodes, or RDF literals such that:
3. the RDF graph \( \mu(\sigma(P)) \) is a subgraph of \( G \).
Evaluation of Basic Graph Patterns

- The result of evaluating $\text{Bgp}(P)$ over $G$ is written $\left[ \text{Bgp}(P) \right]_G$
- The result is a multi set of solutions $\mu$
- The multiplicity of each solution $\mu$ corresponds to the number of different assignments $\sigma$
Multi Sets

**Definition (Multi Set)**

A multi set over a set $S$ is a total function $M : S \rightarrow \mathbb{N}^+ \cup \{\omega\}$

- $\mathbb{N}^+$ denotes the positive natural numbers
- $\omega > n$ for all $n \in \mathbb{N}^+$
- $M(s)$ is the multiplicity of $s \in S$
- $\omega$: countably infinite number of occurrences

- We represent a multi set over the set $S$ also with the set $\{(s, M(s)) \mid s \in S\}$
- We write $(s, n) \in M$ if $M(s) = n$
- We assume that $M(s) = 0$ if $s \notin S$
- Alternative notation: $\{a, b, b\}$ corresponds to the multi set $M$ over the set $\{a, b\}$ with $M(a) = 1$ and $M(b) = 2$
Solution Mapping Example

ex:Birte ex:gives [  
a ex:Lecture ;  
ex:hasTopic "SPARQL" ] .

ex:Sebastian ex:gives [  
a ex:Lecture ;  
ex:hasTopic "DLs and OWL" ] .

Bgp(?who ex:gives _:x .  
_:x ex:hasTopic ?what)
Solution Mapping Example

ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "DLs and OWL" .

Bgp(?who ex:gives _:x .  _:x ex:hasTopic ?what)
Solution Mapping Example

ex:Birte ex:gives _:a .
_ :a rdf:type ex:Lecture .
_ :a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_ :b rdf:type ex:Lecture .
_ :b ex:hasTopic "DLs and OWL" .

Bgp(?who ex:gives _ :x . _ :x ex:hasTopic ?what)

\[ \mu_1 : \ ?who \mapsto \text{ex:Birte,} \quad ?what \mapsto \text{"SPARQL"} \]
\[ \sigma_1 : \ _ :x \mapsto _ :a \]
Solution Mapping Example

ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "DLs and OWL" .

\[ \text{Bgp(} \text{?who ex:gives } _:\text{x} . \text{ } _:\text{x ex:hasTopic } ?\text{what}) \\]

\[ \mu_1 : \text{?who } \mapsto \text{ex:Birte, } \text{?what } \mapsto \text{"SPARQL"} \]
\[ \sigma_1 : \text{ } _:\text{x } \mapsto \text{ } _:\text{a} \]
\[ \mu_2 : \text{?who } \mapsto \text{ex:Sebastian, } \text{?what } \mapsto \text{"DLs and OWL"} \]
\[ \sigma_2 : \text{ } _:\text{x } \mapsto \text{ } _:\text{b} \]
Solution Mapping Example

_ : a rdf: type ex: Lecture .
_ : a ex: hasTopic "SPARQL" .
_ : b rdf: type ex: Lecture .
_ : b ex: hasTopic "DLs and OWL" .

\[
\text{Bgp}(?\text{who ex: gives } _ : x . \ _ : x \ \text{ex: hasTopic } ?\text{what})
\]

\[
\begin{align*}
\mu_1: & \quad ?\text{who } & \rightarrow \text{ex: Birte,} & \quad ?\text{what } & \rightarrow \text{"SPARQL"} \\
\sigma_1: & \quad _ : x & \rightarrow \_ : a \\
\mu_2: & \quad ?\text{who } & \rightarrow \text{ex: Sebastian,} & \quad ?\text{what } & \rightarrow \text{"DLs and OWL"} \\
\sigma_2: & \quad _ : x & \rightarrow \_ : b
\end{align*}
\]

Two solutions each with multiplicity 1
Exercise Solution Sets

ex:Birte ex:gives [  
a ex:Lecture ;  
ex:hasTopic "SPARQL" ] .
ex:Birte ex:gives [  
a ex:Lecture ;  
ex:hasTopic "SPARQL Algebra" ] .

Bgp(?who ex:gives _:_x .  _:_x ex:hasTopic _:_y)
Solution
Solution

ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Birte ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "SPARQL Algebra" .

Bgp(?who ex:gives _:x . _:x ex:hasTopic _:y)
Solution

_ :a rdf:type ex:Lecture .
_ :a ex:hasTopic "SPARQL" .
ex:Birte ex:give _ :b .
_ :b rdf:type ex:Lecture .
_ :b ex:hasTopic "SPARQL Algebra" .

Bgp(?who ex:give _ :x . _ :x ex:hasTopic _ :y)

$\mu_1$: $\mu_1: ?who \mapsto ex: Birte,$

$\sigma_1$: $\sigma_1: _ :x \mapsto _ :a \quad _ :y \mapsto "SPARQL"$
Solution

_ :a rdf:type ex:Lecture .
_ :a ex:hasTopic "SPARQL" .
_ :b rdf:type ex:Lecture .
_ :b ex:hasTopic "SPARQL Algebra" .

Bgp(?who ex:gives _ :x . _ :x ex:hasTopic _ :y)

\[ \mu_1: \quad ?who \mapsto ex:\text{:Birte}, \]
\[ \sigma_1: \quad _ :x \mapsto _ :a \quad _ :y \mapsto "\text{SPARQL}" \]
\[ \mu_2: \quad ?who \mapsto ex:\text{:Birte}, \]
\[ \sigma_2: \quad _ :x \mapsto _ :b \quad _ :y \mapsto "\text{SPARQL Algebra}" \]
Solution

\[
\text{ex:Birte ex:gives _:a .} \\
_:a \text{ rdf:type ex:Lecture .} \\
_:a \text{ ex:hasTopic "SPARQL" .} \\
\text{ex:Birte ex:gives _:b .} \\
_:b \text{ rdf:type ex:Lecture .} \\
_:b \text{ ex:hasTopic "SPARQL Algebra" .}
\]

\[
\text{Bgp( ?who ex:gives _:x . } \_ :x \text{ ex:hasTopic } \_ :y) 
\]

\[
\mu_1: \quad ?\text{who } \mapsto \text{ex:Birte,} \\
\sigma_1: \quad \_ :x \mapsto \_ :a \quad \_ :y \mapsto \text{"SPARQL"} \\
\mu_2: \quad ?\text{who } \mapsto \text{ex:Birte,} \\
\sigma_2: \quad \_ :x \mapsto \_ :b \quad \_ :y \mapsto \text{"SPARQL Algebra"}
\]

One solution with multiplicity 2
Union of Solutions (1)

**Definition (Compatibility)**

Two solutions $\mu_1$ and $\mu_2$ are compatible if $\mu_1(x) = \mu_2(x)$ for all $x$, for which $\mu_1$ and $\mu_2$ are defined.
Union of Solutions (1)

Definition (Compatibility)

Two solutions $\mu_1$ and $\mu_2$ are compatible if
$\mu_1(x) = \mu_2(x)$ for all $x$, for which $\mu_1$ and $\mu_2$ are defined

$\mu_1: ?x \mapsto \text{ex: a}, ?y \mapsto \text{ex: b}$
$\mu_2: ?y \mapsto \text{ex: b}, ?z \mapsto \text{ex: c}$
Union of Solutions (1)

Definition (Compatibility)

Two solutions $\mu_1$ and $\mu_2$ are compatible if $\mu_1(x) = \mu_2(x)$ for all $x$, for which $\mu_1$ and $\mu_2$ are defined.

$\mu_1: \ ?x \mapsto \text{ex} : a, \ ?y \mapsto \text{ex} : b$

$\mu_2: \ ?y \mapsto \text{ex} : b, \ ?z \mapsto \text{ex} : c \ \checkmark$
Union of Solutions (1)

Definition (Compatibility)

Two solutions \( \mu_1 \) and \( \mu_2 \) are compatible if

\( \mu_1(x) = \mu_2(x) \) for all \( x \), for which \( \mu_1 \) and \( \mu_2 \) are defined.

\[
\begin{align*}
\mu_1 &: ?x \mapsto \text{ex: a}, ?y \mapsto \text{ex: b} \\
\mu_2 &: ?y \mapsto \text{ex: b}, ?z \mapsto \text{ex: c} & \checkmark \\
\mu_1 &: ?x \mapsto \text{ex: a}, ?y \mapsto \text{ex: b} \\
\mu_2 &: ?x \mapsto \text{ex: b}, ?z \mapsto \text{ex: c}
\end{align*}
\]
Definition (Compatibility)

Two solutions $\mu_1$ and $\mu_2$ are compatible if $\mu_1(x) = \mu_2(x)$ for all $x$, for which $\mu_1$ and $\mu_2$ are defined.

$\mu_1: ?x \mapsto \text{ex: } a, ?y \mapsto \text{ex: } b$

$\mu_2: ?y \mapsto \text{ex: } b, ?z \mapsto \text{ex: } c \checkmark$

$\mu_1: ?x \mapsto \text{ex: } a, ?y \mapsto \text{ex: } b$

$\mu_2: ?x \mapsto \text{ex: } b, ?z \mapsto \text{ex: } c \not\checkmark$
Union of Solutions (1)

Definition (Compatibility)

Two solutions $\mu_1$ and $\mu_2$ are compatible if $\mu_1(x) = \mu_2(x)$ for all $x$, for which $\mu_1$ and $\mu_2$ are defined.

$\mu_1: ?x \mapsto ex:a, ?y \mapsto ex:b$
$\mu_2: ?y \mapsto ex:b, ?z \mapsto ex:c \checkmark$

$\mu_1: ?x \mapsto ex:a, ?y \mapsto ex:b$
$\mu_2: ?x \mapsto ex:b, ?z \mapsto ex:c \not\checkmark$

$\mu_1: ?x \mapsto ex:a$
$\mu_2: ?y \mapsto ex:b$
Union of Solutions (1)

Definition (Compatibility)

Two solutions $\mu_1$ and $\mu_2$ are compatible if 
$\mu_1(x) = \mu_2(x)$ for all $x$, for which $\mu_1$ and $\mu_2$ are defined

$\mu_1 : ?x \mapsto \text{ex: a}, \ ?y \mapsto \text{ex: b}$
$\mu_2 : ?y \mapsto \text{ex: b}, \ ?z \mapsto \text{ex: c} \quad \checkmark$

$\mu_1 : ?x \mapsto \text{ex: a}, \ ?y \mapsto \text{ex: b}$
$\mu_2 : ?x \mapsto \text{ex: b}, \ ?z \mapsto \text{ex: c} \quad \checkmark$

$\mu_1 : ?x \mapsto \text{ex: a}$
$\mu_2 : ?y \mapsto \text{ex: b} \quad \checkmark$
Union of Solutions (2)

Union of two compatible solutions $\mu_1$ and $\mu_2$:

$$(\mu_1 \cup \mu_2)(x) = \begin{cases} 
\mu_1(x) & \text{if } x \in \text{dom}(\mu_1) \\
\mu_2(x) & \text{otherwise}
\end{cases}$$

⇝ simple intuition: union of matching table rows
Evaluation of Join(·)

For the evaluation of Join($A_1, A_2$) over a graph $G$ with $A_1, A_2$ algebra objects, we define:

- Let $M_1 = [[A_1]]_G$
- Let $M_2 = [[A_2]]_G$
- Let $J(\mu) = \{(\mu_1, \mu_2) \mid M_1(\mu_1) > 0, M_2(\mu_2) > 0,\mu_1$ and $\mu_2$ are compatible and $\mu = \mu_1 \cup \mu_2\}$

$\Rightarrow J$ defines compatible pairs of solutions from $M_1$ and $M_2$

The evaluation $[[\text{Join}(A_1, A_2)]]_G$ results in

$$\left\{(\mu, n) \mid n = \sum_{(\mu_1, \mu_2) \in J(\mu)} (M_1(\mu_1) \times M_2(\mu_2)) > 0\right\}$$
Example to Join(·)

We consider Join($A_1, A_2$) over a graph $G$ with $[A_1]_G = M_1$, $[A_2]_G = M_2$ and:

$M_1 = \{((\mu_1 : ?x \mapsto \text{ex : a}, ?y \mapsto \text{ex : b}), 2),
((\mu_2 : ?x \mapsto \text{ex : a}, 1))\}$

$M_2 = \{((\mu_3 : ?y \mapsto \text{ex : b}, ?z \mapsto \text{ex : c}, 3))\}$

$\mu = ?x \mapsto \text{ex : a}, ?y \mapsto \text{ex : b}, ?z \mapsto \text{ex : c}$

$J(\mu) = \{(\mu_1, \mu_3), (\mu_2, \mu_3)\}$

$\text{Join}(M_1, M_2) = \left\{(\mu, n) \mid n = \sum_{(\mu_1, \mu_2) \in J(\mu)} (M_1(\mu_1) \ast M_2(\mu_2)) > 0\right\}$

$= \{(\mu, 9)\}$

$n = 2 \ast 3 + 1 \ast 3 = 6 + 3 = 9$
Evaluation of Union(·)

The evaluation of Union($A_1, A_2$) over a graph $G$, written $\llbracket\text{Union}(A_1, A_2)\rrbracket_G$, with $A_1, A_2$ algebra objects results in:

\[ \left\{ (\mu, n) \mid M_1 = \llbracket A_1 \rrbracket_G, M_2 = \llbracket A_2 \rrbracket_G, n = M_1(\mu) + M_2(\mu) > 0 \right\} \]
Evaluation of Filter(\cdot)

The evaluation of Filter(F, A) over a graph G, written $\llbracket \text{Filter}(F, A) \rrbracket_G$, with F a filter condition and A an algebra object results in:

$$\{(\mu, n) \mid M = \llbracket A \rrbracket_G, M(\mu) = n > 0 \text{ and } \llbracket \mu(F) \rrbracket = \text{true} \}$$

$\llbracket \mu(F) \rrbracket$ is the Boolean result of evaluating the filter condition.
Evaluation of LeftJoin(\cdot)

The evaluation of $\text{LeftJoin}(A_1, A_2, F)$ over a graph $G$ with $F$ a filter condition and $A_1, A_2$ algebra objects is defined as:

- $M_1 = [A_1]_G$
- $M_2 = [A_2]_G$

The evaluation $[[\text{LeftJoin}(A_1, A_2, F)]]_G$ of $\text{LeftJoin}(A_1, A_2, F)$ over $G$ results in

$$[[\text{Filter}(F, \text{Join}(A_1, A_2))]]_G \cup \left\{ (\mu_1, M_1(\mu_1)) \mid \text{for all } \mu_2 \text{ with } M_2(\mu_2) > 0 : \mu_1 \text{ and } \mu_2 \text{ are incompatible or } [[(\mu_1 \cup \mu_2)(F)] = \text{false} \right\}$$
Example

@prefix ex: <http://eg.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet ex:author ex:Shakespeare ;
ex:price "10.50"^^xsd:decimal .
ex:Macbeth ex:author ex:Shakespeare .
ex:Tamburlaine ex:author ex:Marlowe ;
ex:price "17"^^xsd:integer .
ex:DoctorFaustus ex:author ex:Marlowe ;
ex:price "12"^^xsd:integer ;
ex:title "The Tragical History of Doctor Faustus" .
ex:RomeoJulia ex:author ex:Brooke ;
ex:price "9"^^xsd:integer .

  { ?book ex:author ex:Shakespeare . } UNION 
  { ?book ex:author ex:Marlowe . } 
}

TU Dresden, August 2013 RDF and SPARQL slide 58 of 73
@prefix ex: <http://eg.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet ex:author ex:Shakespeare ;
ex:price "10.50"^^xsd:decimal .
ex:Macbeth ex:author ex:Shakespeare .
ex:Tamburlaine ex:author ex:Marlowe ;
ex:price "17"^^xsd:integer .
ex:DoctorFaustus ex:author ex:Marlowe ;
ex:price "12"^^xsd:integer ;
ex:title
"The Tragical History of Doctor Faustus" .
ex:RomeoJulia ex:author ex:Brooke ;
ex:price "9"^^xsd:integer .

Example Evaluation (1)

\[
\text{Filter(}\ ?\text{price} < 15, \\
\text{Join(} \\
\quad \text{LeftJoin(Bgp(}\ ?\text{book} <\text{http://eg.org/price}> \ ?\text{price}), \\
\quad \text{Bgp(}\ ?\text{book} <\text{http://eg.org/title}> \ ?\text{title}), \\
\quad \text{true),} \\
\quad \text{Union(Bgp(}\ ?\text{book} <\text{http://eg.org/author}> \\
\quad \quad <\text{http://eg.org/Shakespeare}>), \\
\quad \text{Bgp(}\ ?\text{book} <\text{http://eg.org/author}> \\
\quad \quad <\text{http://eg.org/Marlowe}>)))
\]

<table>
<thead>
<tr>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Tamburlaine</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
</tr>
</tbody>
</table>
Example Evaluation (1)

Filter(\(\text{price} < 15\),
\(\text{Join(}
\text{LeftJoin(}\text{Bgp(}\text{?book} <\text{http://eg.org/price}> \text{?price}),
\text{Bgp(}\text{?book} <\text{http://eg.org/title}> \text{?title}),
\text{true}),
\text{Union(}\text{Bgp(}\text{?book} <\text{http://eg.org/author}>\text{<http://eg.org/Shakespeare}>),
\text{Bgp(}\text{?book} <\text{http://eg.org/author}>\text{<http://eg.org/Marlowe}>))))

<table>
<thead>
<tr>
<th>book</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Tamburlaine</td>
<td>ex:Macbeth</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>ex:Hamlet</td>
</tr>
</tbody>
</table>
Example Evaluation (2)

Filter(?price < 15,
    Join(
        LeftJoin(Bgp(?book <http://eg.org/price> ?price),
                true),
        Union(Bgp(?book <http://eg.org/author>
                   <http://eg.org/Shakespeare>),
               Bgp(?book <http://eg.org/author>
                   <http://eg.org/Marlowe>))))

<table>
<thead>
<tr>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
</tr>
<tr>
<td>ex:Macbeth</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
</tr>
</tbody>
</table>
Example Evaluation (3)

Filter(?
price < 15,
Join(  
  LeftJoin(Bgp(?
  Bgp(?
  true),
  Union(Bgp(?
book <http://eg.org/author>  
  <http://eg.org/Shakespeare>),  
  Bgp(?
book <http://eg.org/author>  
  <http://eg.org/Marlowe>))))

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
</tr>
</tbody>
</table>
Example Evaluation (3)

Filter(?price < 15, Join(

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:DoctorFaustus</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
Example Evaluation (4)

Filter(?price < 15, 
   Join( 
      LeftJoin(Bgp(?book <http://eg.org/price> ?price), 
              true), 
   Union(Bgp(?book <http://eg.org/author> 
           <http://eg.org/Shakespeare>), 
         Bgp(?book <http://eg.org/author> 
             <http://eg.org/Marlowe>))))

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Example Evaluation (5)

Filter(?price < 15,
   Join(
      LeftJoin(Bgp(?book <http://eg.org/price> ?price),
               true),
      Union(Bgp(?book <http://eg.org/author>
                        <http://eg.org/Shakespeare>),
               Bgp(?book <http://eg.org/author>
                        <http://eg.org/Marlowe>))))

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
Example Evaluation (6)

Filter(?price < 15,
    Join(
        LeftJoin(Bgp(?book <http://eg.org/price> ?price),
            true),
        Union(Bgp(?book <http://eg.org/author> <http://eg.org/Shakespeare>),

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
Agenda

1 Recap
2 SPARQL Semantics
3 Transformation of Queries into Algebra Objects
4 Evaluation of the SPARQL Algebra
5 Operators for the Modifiers
6 Summary
### Operators for Representing the Modifiers

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ToList(M)</code></td>
<td>Constructs from a multi set a sequence with the same elements and multiplicity (arbitrary order, duplicates not necessarily adjacent)</td>
</tr>
<tr>
<td><code>OrderBy(M, comparators)</code></td>
<td>Sorts the solutions</td>
</tr>
<tr>
<td><code>Distinct(M)</code></td>
<td>Removes the duplicates</td>
</tr>
<tr>
<td><code>Reduced(M)</code></td>
<td>May remove duplicates</td>
</tr>
<tr>
<td><code>Slice(M, o, l)</code></td>
<td>Cuts the solutions to a list of length l starting from position o</td>
</tr>
<tr>
<td><code>Project(M, vars)</code></td>
<td>Projects out the mentioned variables</td>
</tr>
</tbody>
</table>
Transformation of the Modifiers

Let $q$ be a SPARQL query with pattern $P$ and corresponding algebra object $A_P$. We construct an algebra object $A_q$ for $q$ as follows:

1. $A_q := \text{ToList}(A_P)$
2. $A_q := \text{OrderBy}(A_q, (c_1, \ldots, c_n))$ if $q$ contains an ORDER BY clause with comparators $c_1, \ldots, c_n$
3. $A_q := \text{Project}(A_q, \text{vars})$ if the result format is SELECT with vars the selected variables (\* all variables in scope)
4. $A_q := \text{Distinct}(A_q)$ is the result format is SELECT and $q$ contains DISTINCT
5. $A_q := \text{Reduced}(A_q)$ if the result format is SELECT and $q$ contains REDUCED
6. $A_q := \text{Slice}(A_q, \text{start}, \text{length})$ if the query contains OFFSET start or LIMIT length where start defaults to 0 and length defaults to $(|A_q| - \text{start})$
Evaluation of the Modifiers

The algebra objects for the modifiers are recursively evaluated

- Evaluate the algebra expression of the operator
- Apply the operations for the solution modifiers to the obtained solutions
Agenda

1 Recap
2 SPARQL Semantics
3 Transformation of Queries into Algebra Objects
4 Evaluation of the SPARQL Algebra
5 Operators for the Modifiers
6 Summary
Summary

- We learned how to evaluate SPARQL queries
- The query is transformed into an algebra object
- The query basic graph patterns generate solutions
- The other operators combine solutions
- The result format determines how the solutions are presented